So long, and thanks for all the tests

Seb Rose
Claysnow Limited
@sebrose
DOUGLAS ADAMS
SO LONG, AND THANKS FOR ALL THE FISH
“The last ever dolphin message was misinterpreted as a surprisingly sophisticated attempt to do a double-backwards-somersault through a hoop whilst whistling the 'Star Spangled Banner', but in fact the message was this: So long and thanks for all the fish.”

Why do we test software?

“a process of gathering information about [software] with the intent that the information could be used for some purpose”
Developer tests

What are they for?
• making sure the code works
• preventing regressions
• documenting the code
• driving a testable design
“I get paid for code that works, not for tests, so my philosophy is to test as little as possible to reach a given level of confidence ... 

“I suspect this level of confidence is high compared to industry standards”

http://stackoverflow.com/questions/153234/how-deep-are-your-unit-tests/153565#153565
Write a failing test → Refactor → Make the test pass
Developer testing

When do you write tests?
• never
• after writing the code
• while you write the code
• before writing the code
Any objections?

http://www.slideshare.net/sebrose/common-objections-to-tdd-and-their-refutations
“Takes too much time to write tests.”

“If I change something it will break a whole bunch of tests that I will have to fix and it will be more work for me in the end than just verifying my changes manually.”

“Running the tests takes too long.”

http://www.slideshare.net/sebrose/common-objections-to-tdd-and-their-refutations
“No scientific proof of it actually having any benefit (compared to just code reviews, pair programming or etc.) for the same amount of time”

“It's not practical for the kind of work I do. By which I mean, it cannot be conclusively shown to provide tangible benefits that outweigh the perceived costs.”

http://www.slideshare.net/sebrose/common-objections-to-tdd-and-their-refutations
TDD - Where did it all go wrong?

Ian Cooper - http://vimeo.com/68375232

Why are people abandoning TDD?

- Test first is unproductive obstacle
- Faster feedback gives the competitive advantage
- It’s difficult to understand the test’s intent
Isn’t agility about speed?
- Fast feedback
- Understand the risk
- Use appropriate methods
TDD is dead

David Heinemeier Hansson-
http://david.heinemeierhansson.com/2014/tdd-is-dead-long-live-testing.html
http://david.heinemeierhansson.com/2014/test-induced-design-damage.html

TDD isn’t useful anymore
- We’ve learnt what we needed
- Unit tests aren’t useful
- Testability hurts the design
Oh no it’s not

Robert C. Martin (aka Uncle Bob) -
http://blog.8thlight.com/uncle-bob/2014/04/25/MonogamousTDD.html
http://blog.8thlight.com/uncle-bob/2014/04/30/When-tdd-does-not-work.html
http://blog.8thlight.com/uncle-bob/2014/05/01/Design-Damage.html

TDD is still useful

- Unit tests are fast and thorough
- It’s not universally applicable
- Decomposition is good design
Given a process B, which follows a process A, sometimes in performing B we need to perform some of A again. We can remove the need to rework by taking some portion of process B and performing it before process A.
Emergent design

“With emergent design, a development organization starts delivering functionality and lets the design emerge.

...while it is being built, the organization will [...] refactor out the commonality ...

At the end [we are] left with the smallest set of the design needed, as opposed to the design that could have been anticipated in advance.”

Adaptable design up-front?

“... software systems must evolve, and that this evolution must be supported through well planned, adaptable designs.”

http://effectivesoftwaredesign.com/2013/10/18/avoiding-technical-debt-how-to-accumulate-technical-savings/
Fake it until you make it

“The goal of this game is to discover as many collaboration objects as necessary in order to define leaf nodes that implement a piece of narrowly-defined logic that your feature needs.”

Domain discontinuity

Robert C. Martin (aka Uncle Bob) -
http://blog.8thlight.com/uncle-bob/2014/01/27/TheChickenOrTheRoad.html

“First I am going to express my deep disagreement with the Author.

I will refute his arguments and utterly destroy his conclusions.

And then, once I'm done salting the ground where he used to live, I'll tell you why I completely agree with him.”
Domain discontinuity

Robert C. Martin (aka Uncle Bob) -

http://blog.8thlight.com/uncle-bob/2014/01/27/TheChickenOrTheRoad.html

TDD works within an architectural domain

- Refactoring is cheap
- Test the behaviour not the implementation
- We won’t get it right the first time
Domain discontinuity

Robert C. Martin (aka Uncle Bob) -
http://blog.8thlight.com/uncle-bob/2014/01/27/TheChickenOrTheRoad.html

TDD doesn’t work across domains

- Refactoring is costly
- Moving behaviours requires new tests
- Architecture must be up-front
What about architecture and design?

https://leanpub.com/software-architecture-for-developers
Dreyfus model of skill acquisition

from the best-selling authors of PEOPLEWARE

Waltzing with Bears
MANAGING RISK ON SOFTWARE PROJECTS

"The seminal work on managing software project risk... Explosive insights, practical advice. Finally we have a guide to risk management that we can implement and use."
—Rob Austin, Professor Harvard Business School

Tom DeMarco & Timothy Lister
Where should we test?

http://www.slideshare.net/dannorth/accelerating-agile-testing
Where should we test?

http://www.slideshare.net/dannorth/accelerating-agile-testing
But, Dan. What about the Extreme Rules?

Code may not be released unless it:

- is formatted to agreed CodingConventions
- has been coded by PairProgramming.
- has UnitTests
- has passed all UnitTests at 100%.

http://c2.com/cgi/wiki?ExtremeRules
There’s a rule to cover that...

Theyre Just Rules
One of the many ExtremeRules.

If a rule isn't working, the team can remove it from the rule list.

http://c2.com/cgi/wiki?TheyreJustRules
Which test first?

Inside-out? - “Classicist”

Outside-in? - “London school”

In the middle? - also “Classic”

http://martinfowler.com/articles/mocksArentStubs.html
http://programmers.stackexchange.com/questions/166409/tdd-outside-in-vs-inside-out
Write a failing end-to-end test

Write a failing unit test

Make the test pass

Refactor

Nat Pryce
What is a unit test anyway?

A test is not a unit test if:

• It talks to the database
• It communicates across the network
• It touches the file system
• It can’t run at the same time as other unit tests
• You have to do special things to your environment (such as editing config files) to run it

http://www.artima.com/weblogs/viewpost.jsp?thread=126923
Test behaviours not implementation details

“When we refactor we don’t want to break tests. If our tests know too much about our implementation, that will be difficult, because changes to our implementation will necessarily result in us re-writing tests – at which point we are not refactoring. We would say that we have over-specified through our tests.

Instead of assisting change, our tests have now begun to hamper it.”

http://vimeo.com/68375232
http://codebetter.com/iancooper/2011/10/06/avoid-testing-implementation-details-test-behaviours/
Properties of unit testing

Understandable
Maintainable
Repeatable
Necessary
Granular
Fast
How can we test a component in isolation?

Design for testability

- Inversion of control
- Dependency Injection

http://www.jamesshore.com/Blog/Continuous-Integration-on-a-Dollar-a-Day.html
Mocking in a nutshell
The Meszaros taxonomy

- **Dummy** objects are passed around but never actually used.
- **Fake** objects have working implementations.
- **Stubs** provide canned responses.
- **Mocks** are pre-programmed with expectations which form a specification of the calls they are expected to receive.

http://xunitpatterns.com
Mocking can give you a false sense of security
Mocking can make a terrible mess
Contract and Collaboration tests

- A stub in a collaboration test must correspond to an expected result in a contract test
- An expectation in a collaboration test must correspond to an action in a contract test

This provides a systematic way to check that [your code] remains in sync with the implementations of [the component you are mocking].

via JB Rainsberger, GOOS mailing list, “Unit-test mock/stub assumptions rots”, 15 March 2012
1. Change the implementation \( A_1 \) of \( A \), \textit{noticing} a change in the contract of \( A \).

2. For \textit{each} change in the contract of \( A \):

   1. If an action in \( A \) has changed (parameter changed, method name changed), then look for expectations of that action in collaboration tests, and change them to \textit{match} the new action.

   2. If a response from \( A \) has changed (return type, value, what the value means), then look for stubs of that action in collaboration tests, and change them to \textit{match} the new response.

via JB Rainsberger, GOOS mailing list, “Unit-test mock/stub assumptions rots”, 15 March 2012
Clean Code
A Handbook of Agile Software Craftsmanship

Foreword by James O. Coplien

Robert C. Martin
Are we really writing tests?

“If somehow all your production code got deleted, but you had a backup of your tests, then you'd be able to recreate the production system with a little work.

“If, however, it was your tests that got deleted, then you'd have no tests to keep the production code clean. The production code would inevitably rot, slowing you down.”

Or are we writing specifications?

“So we can conclude that if it became a choice between the tests or the production code, we'd rather preserve the tests.

“And this means that the tests are a more important component of the system than the production code is.

“Because the tests are the specs.”

What use is coverage?
Mutation testing

Real world mutation testing
PIT is a state of the art mutation testing system, providing gold standard test coverage for Java and the JVM. It’s fast, scalable and integrates with modern test and build tooling.

http://pitest.org
cyber-dojo
a place to practice programming

get a new practice-id
setup

enter a practice-id

start  resume  review

If you find cyber-dojo useful please make a donation. All donations are used solely to buy raspberry pi computers for children. Thank you.
faqs

Why don't you add syntax highlighting?

No. Listen. Stop trying to go faster, start trying to go slower. Don't think about finishing, think about improving. Think about practising as a team. That's what cyber-dojo is built for.

Why don't you add auto-refactoring?

No. Listen. Stop trying to go faster, start trying to go slower. Don't think about finishing, think about improving. Think about practising as a team. That's what cyber-dojo is built for.

Why don't you add ...?

No. Listen. Stop trying to go faster, start trying to go slower. Don't think about finishing, think about improving. Think about practising as a team. That's what cyber-dojo is built for.
Things that seem important to me:

- chase the VALUE
- identify the RISKS
- understand your CONTEXT

The tests should make your life EASIER
Seb Rose

Twitter: @sebrose
Blog: www.claysnow.co.uk
E-mail: seb@claysnow.co.uk
Backup
Step 1

Write exactly one new test. It should be the smallest test which seems to point in the direction of a solution

Step 2

Run the test to make sure it fails

TDD as if you meant it

Step 3

Make the test from (Step 1) pass by writing the least amount of implementation code you can IN THE TEST METHOD.

Step 4

Refactor to remove duplication or otherwise as required to improve the design. Be strict about the refactorings. Only introduce new abstractions (methods, classes, etc) when they will help to improve the design of the code. Specifically...

TDD as if you meant it

Step 4 (continued)

1. ONLY Extract a new method if there is sufficient code duplication in the test methods. When extracting a method, initially extract it to the test class (don't create a new class yet).

2. ONLY create a new class when a clear grouping of methods emerges and when the test class starts to feel crowded or too large.

Step 5

Repeat the process by writing another test (go back to Step 1).